# Laboratory 2 Introduction to Assembly Programming

Last Revised: May 2000

## *1* Objectives

With the completion of this lab, you should be able to:

- Assemble, link, download, and run machine-language programs
- Use MASM to find syntax errors
- Use MUN-88 Monitor Commands to debug logic errors
- Write some simple 8086/8088 assembly language programs

## *2* Introduction

In this lab, you will write assembler code to perform certain simple I/O operations, in order to gain familiarity with the Intel 8086/8088 software. You will then assemble, link and download the machine code to the MUN-88 single board computer, and finally test your programs.

You do not need to submit any comments or code for section 3; you do for sections 4 & 5.

## *3* Assemble, Link, Convert, Download

Using a text editor, such as MS-DOS *EDIT* or Windows *Notepad*, create a new file called ***lab2a.asm*** and enter the following code into that file:

```
            TITLE  hello
TEXT        SEGMENT
            ASSUME cs:TEXT,ds:TEXT,es:TEXT

start:      mov    ax, cs     ; Always start with load ds,es
            mov    ds, ax
            mov    es, ax
; This code is the same as that used in the previous lab
            mov    cx, 8
            mov    al, 11111110b
do_it:      out    30h, al
            rol    al, 1
            call   delay_10ms
            loop   do_it
            int    6          ; Return to monitor

delay_10ms: mov    dx, 2000h  ; Delay subroutine
rep1:       dec    dx
            jnz    rep1
            ret
; The following two lines and the first three contain assembler
directives
TEXT        ENDS              ; End Segment
            END     start
```

Before proceeding further, notice a few important points in the above code. Microsoft assembler requires that you include these first three lines and the last two lines in any 8086/8088 assembler program for successful assembly. There are several possible variations of these *assembler directives*, but they are not important for these simple programs. The *TITLE* of the program can be anything that you choose. There should be a label at the beginning of the assembler code (here, ***start***), and this label should be specified after the last *END*. The last instruction executed in your program should be ***int 6***. This hands control back to the MUN-88 monitor upon completion of the program execution. Any other instruction may cause the system to crash.

## 3-1 Assembling & Linking

You should now assemble the program with the Microsoft assembler. If you haven't already done so, open an MS-DOS window using the Windows NT sequence **Start -> Program Files -> MS-DOS prompt for MUN-88**. This is important because a number of necessary directories are added to the command path.

At the DOS prompt, switch directories so that you are in the same directory as where you saved the above file. When you type the command **masm lab2a**, you will encounter an exchange similar to the following. The commands are in bold.

```
M:> masm lab2a
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.
Object filename [lab2a.OBJ]: (press ENTER)
Source listing  [NUL.LST]: lab2a
Cross-reference [NUL.CRF]: (press ENTER)
  47752 + 410931 Bytes symbol space free
      0 Warning Errors
      0 Severe  Errors
M:>
```

The assembler first produces an object file containing the machine code of your program (**lab2a.obj**). The assembler produces this file by default, and thus you do not need to type in the full name.

The next file is a list file. The assembler does not create this file automatically, so you must type in a name for it (in this case, **lab2a**, or **lab2a.lst**). If there are any syntactic errors in your code, the assembler reports it through *severe errors* and *warnings*. The list file would help you pin down the location and the probable cause of any such error. The list file also shows the actual machine code generated by the assembler. If there are errors, fix them in the assembler file and run the assembler again.

You don't need the cross-reference file, so press ***ENTER*** to skip its generation.

For the purposes of submission, print out the *list* file. (If you use Windows to print, you may save paper by printing *2 up* per page – see the layout section of *Print Properties* dialog).

After successful assembly, invoke the linker by typing in following commands:

```
M:>link lab2a
```

```
Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Run File [lab2a.EXE]: (press ENTER)
List File [NUL.MAP]: (press ENTER)
Libraries [.LIB]: (press ENTER)
LINK : warning L4021: no stack segment
M:>
```

Ignore the linker warning about a missing stack segment. The linking process results in the creation of an executable file named **lab2a.exe**. In order to create a file that will run on the MUN-88, convert the executable file into a binary file by using the following DOS command:

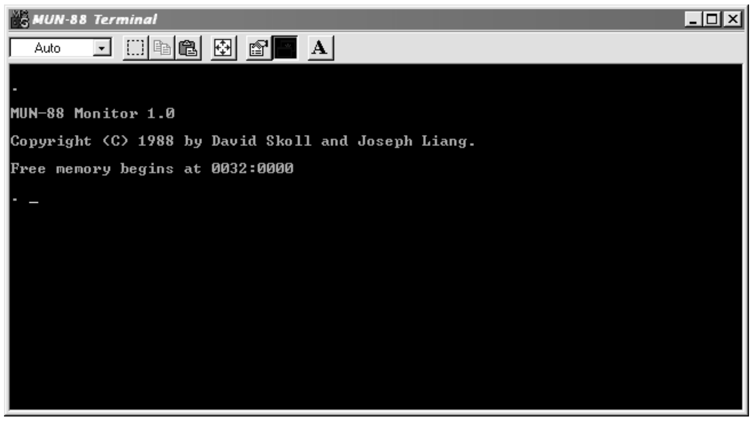
```
M:>exe2bin lab2a
```

The binary file *lab2a.bin* is now ready for downloading to the MUN-88 board.

## 3-2 Downloading

You will not be using Kermit to communicate with MUN-88, but a new program. It is called *Terminal*, and is located on your ***local*** drive, under **mun88\software\**. This drive should automatically be mapped (probably to drive **L:**) when you log on to a Windows NT computer in the lab. Use the Windows Explorer to open the directory **L:\mun88\software\**, and double-click on **Terminal.exe** to start it.

You will then be presented with two windows – a terminal window (see Figure 2-1) and a download window (see Figure 2-2).



```
.
MUN-88 Monitor 1.0
Copyright (C) 1988 by David Skoll and Joseph Liang.
Free memory begins at 0032:0000
. _
```

**Figure 2-1: The main MUN-88 Terminal window**

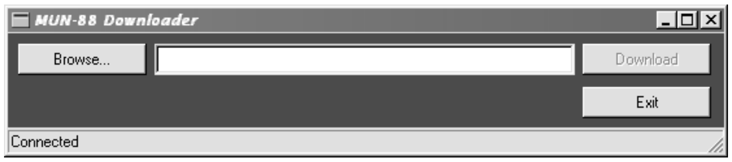


**Figure 2-2: The MUN-88 Downloader window**

If the MUN-88 board is turned on, you will automatically be connected to it. The status bar in the ***MUN-88 Downloader*** window will indicate if you have been successful in connecting. If it reports “Connected”, then you should see the dot-prompt of the MUN-88 Monitor in the ***MUN-88 Terminal*** window. If instead you see another character, such as an up arrow, then the baud rate of the board is incorrect. Switch the first 3 DIP switches off, and reset the board. You should then see the MUN-88 welcome message.

Free memory in the MUN-88 starts at location 0032H:0000H, or physical location 00320H. Since the board has 2KB of RAM, you may store your generated machine code anywhere between 00320H and 007FFH. Thus you may store multiple programs at different locations in RAM at the same time. As long as you do not turn off the power, the code should be stored safely.

For this program, store it starting at the free memory location. Invoke the download command in the ***MUN-88 Terminal*** by typing:

```
. DL 32:0
Ready to download...
```

You should then see the “Ready to download…” message displayed, and **no** dot-prompt. The MUN-88 is now in a special state where it expects to receive machine code bytes on the serial connection.

Activate the ***MUN-88 Downloader*** window by selecting it with the mouse. Click on the **Browse…** button to select your binary file. After selecting the file, the full name and path will appear in the ***MUN-88 Downloader*** window, and the **Download** button will become selectable. Click on the button to download the code. You should quickly see the message in the status bar change to a message saying “Sent 29 bytes on COM2”, and the ***MUN-88 Terminal*** window should have the message “Download OK.”

## 3-3 Executing

Switch back to the ***MUN-88 Terminal*** window by clicking on it. Run your program.

```
. GO 32:0
Program Terminated at 0032:0014


.
```

You should see the same pattern on the LEDs as in Lab 1. Congratulations on creating your first assembly language program!

## *4* Debugging

Gaining experience with debugging is very important; if a program does not work, you need to be able to determine why. Simply looking at the ASM file you wrote often does not work. In this part of the lab, you will use several debugging features available to you in MASM and in the MUN-88 Monitor.

Use a web browser to go to **http://www.engr.mun.ca/~charlesr/eng4862/labs**, and download the file called **lab2b.asm** to your directory. You can do this by right-clicking on the name, and selecting **Save As…** in the pop-up menu that appears. You will use this file to examine some of the syntax reporting mechanisms of MASM, and to try out the debugging facilities provided by the MUN-88 Monitor.

Look at the ASM file using a text editor. Describe what this program is supposed to do.

### 4-1 Syntax errors and the LST file

The list file generated by MASM is very useful for finding syntax errors in your code. This includes misspelled and invalid assembly mnemonics, as well as mistakes with labels and operands. The **lab2b.asm** file has several errors.

Using an MS-DOS window, run MASM on the **lab2b.asm** file. Make sure you generate a list (**.LST**) file.

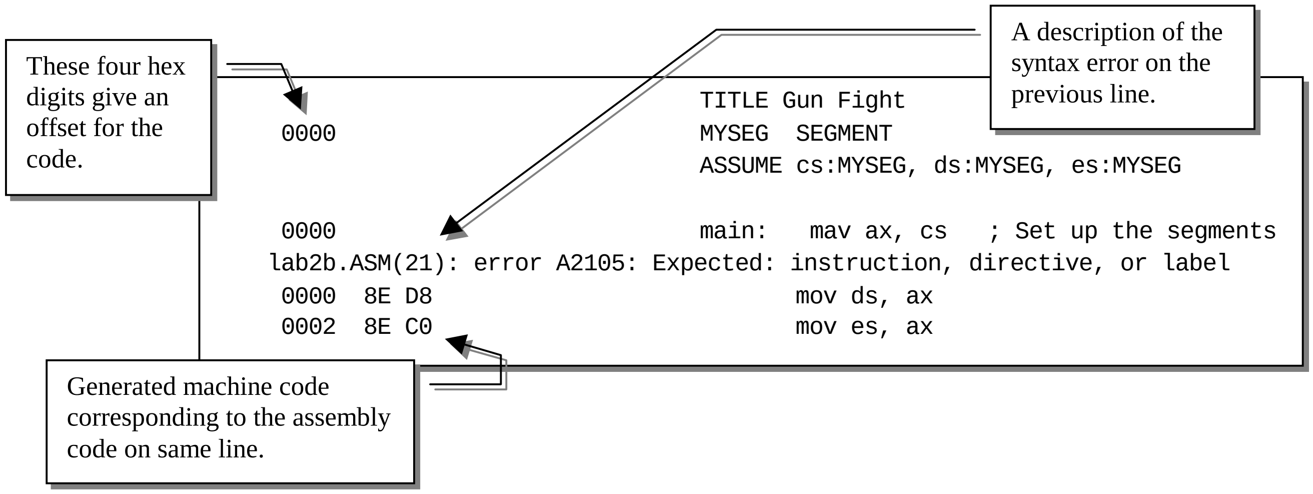
You should receive 10 severe errors, as follows:

```
lab2b.ASM(21): error A2105: Expected: instruction, directive, or label
lab2b.ASM(25): error A2009: Symbol not defined: FFH
lab2b.ASM(39): error A2050: Value out of range
lab2b.ASM(46): error A2029: Division by 0 or overflow
lab2b.ASM(53): error A2029: Division by 0 or overflow
lab2b.ASM(57): error A2029: Division by 0 or overflow
lab2b.ASM(61): error A2029: Division by 0 or overflow
lab2b.ASM(65): error A2029: Division by 0 or overflow
lab2b.ASM(69): error A2029: Division by 0 or overflow
lab2b.ASM(73): error A2029: Division by 0 or overflow

  47764 + 410903 Bytes symbol space free

      0 Warning Errors
     10 Severe  Errors
```

You must fix these errors before the assembler will create an object file. Open up the list file in a text editor to see the location of the errors, and an indication of how to fix them. The following shows a portion of the resulting list file.


These four hex digits give an offset for the code.
A description of the syntax error on the previous line.
Generated machine code corresponding to the assembly code on same line.

```
                                TITLE Gun Fight
 0000                           MYSEG  SEGMENT
                                ASSUME cs:MYSEG, ds:MYSEG, es:MYSEG

 0000                           main:   mav ax, cs    ; Set up the segments
lab2b.ASM(21): error A2105: Expected: instruction, directive, or label
 0000  8E D8                            mov ds, ax
 0002  8E C0                            mov es, ax
```

The first error is located after what should be the first line of code. No machine code was generated because the instruction is not spelled correctly. Now that you know the problem, fix the instruction **in the original ASM file**.

Go through the rest of the list file. What are each of the errors? Fix all ten errors, and try running MASM again. You should be able to successfully assemble the file. Link the OBJ file, and create the BIN file. Use the Terminal program to download the BIN file to the MUN-88. The ***MUN-88 Downloader*** should report that it has sent 80 bytes.

## 4-2 Single-Step

Now that the file is on the MUN-88, try to execute it. Describe what happens. Is this expected? Try toggling DIP switches 1 or 8. Does the program stop?

If you have to, press the reset button. Turn on the Monitor single-step mode by issuing the correct command at the MUN-88 Monitor dot-prompt. When you start the program now, you should see a report similar to the following:

```
. go 32:0

AX=0032 BX=00FF CX=0000 DX=0000 SI=0000 DI=0000 SP=0310 BP=0000
CS=0032 DS=0032 ES=0032 SS=0000 IP=0002 O=0 D=0 I=0 T=1 S=0 Z=0 A=0 P=0 C=0

0032:0002 8E

.
```

The Monitor executes the first instruction, and then halts the program. The contents of all the registers are displayed; their values reflect their state after the first instruction has completed. Thus you can see that the first instruction of the program (**MOV AX,CS**) has completed successfully.

The line “0032:0002 8E” shows the location and first byte of the next instruction to be executed. What is this instruction?

Using the list file, determine and state the offset of the first **IN** instruction. If you issue **go** without a segment and offset, the instruction uses the **CS** and **IP** registers to determine the next instruction. Issue the **go** several times, until the **IN** instruction completes execution. Based on the value in the **AL** register, has the **IN** instruction worked correctly? What value is in the **AL** register? What value did you expect?

Use the list file to determine what the problem is. Does this problem occur elsewhere in the code? Fix the problem in the ASM file, then assemble, link, convert, and download to the MUN-88 board.

Turn off single step mode, and run the program. Has this fixed all of the errors? Does switching DIP number 1 or 8 stop execution? What happens to the LEDs when you move one of the switches?

## 4-3 Breakpoints

Until you switch DIP number 1 or 8, the program will likely run for a very long time before the LEDs will stop blinking. For some reason, the program is not finishing the double loop where the lights blink.

Using the list file, determine the offsets of the instructions **dec dx** and **dec cx**. Using MUN-88 Monitor commands, set a breakpoint at each of these offsets. Type **go 32:0**; at what breakpoint does the program stop executing? To which instruction does this correspond? Report the value of register that will be modified, and type **go**. What happens? Type **go** a few more times, and report the results. Does the program seem to be working as expected at this point?

To reach the other breakpoint, you will have to disable only the first breakpoint by using the correct Monitor command. Do this, and type **go**. What instruction will be executed the next time you type go? What is the value of the corresponding register? Type **go** a few more times, and comment on the results. When will the LEDs finally stop blinking?

Based on what you have found out from the breakpoints, fix the ASM file by loading the appropriate register with a value before the line labeled *rep2*. A value between 20 and 40 is recommended. Assemble, link, convert and download this file.

Clear the breakpoints, and use the new list file to set a breakpoint at the previously offending **dec** instruction. Run the program again, and check that the program is now running correctly.

Clear all breakpoints and run the program. Test the correctness of the program by switching both DIPs number 1 and 8 before and after the LEDs have stopped blinking. Briefly describe what happens. Then challenge some classmates to a duel.

Print out the list file of your final, correct version of the program. You might wish to remove the symbols at the top of the file, which are printer commands to start a new page. Indicate on the listing all of the changes you have made.

**Optional:**

You can use the list file to determine the exact location, in your machine code, of the number of blinks. Using the MUN-88 Monitor command modify, you can change the number of blinks without re-assembling your code. Try it out!

## *5* Programming

This section of the lab requires you to write two programs in assembly language, assemble them using MASM, and link and convert them using ***exe2bin***.

Use the following code as a base for your programs. Replace the words in bold with the appropriate text.

```
            TITLE   your title here
MYSEG       SEGMENT
            ASSUME cs:MYSEG, ds:MYSEG, es:MYSEG

main:       mov    ax, cs      ; Always start with load ds,es
            mov    ds, ax
            mov    es, ax

            code
      ; Apply comments liberally
            more code

            int 6              ; return to Monitor

MYSEG       ENDS               ; End Segment
            END   main
```

At the end of your main code body, use the **int 6** instruction to return control to the MUN-88 Monitor.

### 5-1 Program 1

Write an assembler program to read the DIP switches and to load the DH register with this result. Then display the value on the DIP switches on the LEDs. A switch that is ON should light the corresponding light in the LEDs.

Assemble, link, download and test proper execution of your program for various switch settings. Print a copy of your ***LST file***, and demonstrate your program to a TA. Make sure you place appropriate comments **in** your code.

### 5-2 Program 2

Write a program to light up the odd numbered lights (as numbered on the board), wait about 1 second, and then light up the even numbered lights. The program must then wait for the DIP switch at bit 3 (**not** switch number 3!) to change position. The program should then clear the lights, and terminate.

## *6* Submission

At the end of the lab submit print-outs of the three ***list files*** with handwritten comments on the success of each program. Save copies of all your programs.